

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

**APPLICATION FOR UNITED STATES PATENT**

**FOR**

**DATA REPLICATION FACILITY FOR DISTRIBUTED COMPUTING  
ENVIRONMENTS**

**Inventors: Richard J. Nordin; Andreas L. Bauer; Sriram Krishnan; Gregory W. Lazar**

**Attorneys:  
Joel Wall, Esquire  
P.O. Box 169  
Hopkinton, MA 01748  
508-435-4432  
December 19, 2001**

# DATA REPLICATION FACILITY FOR DISTRIBUTED COMPUTING ENVIRONMENTS

## BACKGROUND OF THE INVENTION

### 1. Field of the Invention:

The present invention relates to apparatus, methodology, systems, and/or computer program product for persisting or replicating data in a distributed computing environment and, more particularly, relates to a file replication service (FRS) which utilizes a directory provider that controls a directory data base (DDB) distributed throughout a client-server computer storage network in combination with both file version variable observers, also distributed throughout the network, and syncing threads utilizable by each node in the network.

### 2. Description of Prior Art:

Computer networks are having an ever-increasing impact on modern-day lifestyle for many people, as the Internet, for example, is demonstrating. The Internet is supported by a technological infrastructure including other networks which can also be utilized in non-Internet environments. These other computer networks, for example client-server storage networks, may not be as well known to the general public as the Internet, but nevertheless can provide very important data storage and retrieval functions in a wide variety of applications (e.g. banking and finance, traffic control, medical research, military applications, routine business, etc.) to the overall benefit of society. In a client-

1 server storage network, a human user of the network can be conceptualized as the  
2 “client” who is being “served” by the network. And, within such a network, inanimate  
3 hardware/software sub-systems that are being “served” in some capacity by other such  
4 subsystems (“servers”) are also referred to as “clients” of those servers.

5  
6 A client-server network today may typically be based on an object oriented  
7 computer system that employs one or more object-oriented computer languages such as  
8 C++, XML (eXtensible Markup Language), JAVA, and/or others. Briefly, an object, in  
9 computer software terms, is a dedicated area of memory which can be thought of as an  
10 impervious container holding both data and instructions within itself, both defining itself  
11 and its relationships to other objects in the computer system or network. Such object or  
12 node can send and receive messages to and from other objects, respond and react to such  
13 messages (e.g. commands) but shall normally be impervious to internal scrutiny. For  
14 example, in a computer data storage system (a kind of computer) each object (system  
15 object) may describe or relate to a specific tangible detail in the storage system or in the  
16 storage system’s processor (e.g., details such as those describing or relating to aspects of  
17 operation of the processor’s cooling-fan, power switch, cache memory, power supply,  
18 disk drive interface, etc.). These tangible objects (nodes) in the storage system can send  
19 messages to each other within the storage system and to other objects outside the storage  
20 system over the network with which they are operatively coupled. Also, the storage  
21 system itself can be an object and interact as a node with other nodes in a network.  
22 Storage system or other kinds of objects that do not have special privileges relative to  
23 each other are sometimes termed “slave” nodes. By comparison, a “master” node can

1 have certain leadership or control privileges or responsibilities relative to slave nodes in  
2 its network. Also, within the storage system, a segregated amount of data, or a file of  
3 data, or a data structure, or parts of the file such as a hash number or other data  
4 representation or tag such as a file version number, can also be thought of, and treated as  
5 an object.

6  
7 In a client-server computer data storage network, its principal purpose is to store  
8 and retrieve data in an efficient, accurate and reliable manner under a wide variety of  
9 conditions imposed on the network. Under certain circumstances, particular data  
10 introduced to a storage network is to be stored not only on one of the network's nodes or  
11 storage systems, but is to be stored on all of them. For example, if there are a number of  
12 human users interacting with a network (possibly globally linked via the Internet) having  
13 storage system nodes located worldwide, it is important for each user to have a current  
14 list of all authorized network users. In this case, this security data needs to be stored on  
15 each network storage system node worldwide, and needs to be updated on all nodes if and  
16 when data changes (when a new user is added or prior user is dropped).

17  
18 In the prior art, a technique for accomplishing storage of the same particular data  
19 on each of a number of network nodes is offered by Microsoft Inc. and is known as  
20 "Active Directory". Among other drawbacks and differences from the present invention,  
21 this offering requires the network's global administrator to specify network topology.  
22 This is normally a complicated, cumbersome, and time-consuming task of network  
23 topological configuration for the administrator to undertake. Other prior art offerings

1 include, for example, the placing of a file server into the network or domain, where  
2 network nodes seek updated file copies from the file server but have drawbacks such as  
3 vulnerability to a single point of failure. In this instance, the data is kept in a centralized  
4 place, such as a shared directory, which therefore impacts all other nodes if and when the  
5 directory becomes inaccessible. Accordingly, if the single vulnerable link fails (file  
6 server failure or shared directory inaccessibility) the entire network cannot be updated  
7 and therefore fails catastrophically.

8  
9 Thus, with respect to new or updated data files that are introduced into a client  
10 server network, there is a need for a service to replicate or duplicate certain of them in  
11 each storage system or node in the network in a manner that both avoids both topological  
12 constraints and single point of failure designs. Embodiments of the present invention  
13 satisfy this need and are welcome solutions to these problems and shortcomings of the  
14 prior art. Embodiments of the present invention include a Directory Provider Service  
15 (DPS) which controls, among other things, a Directory Data Base (DDB) distributed  
16 throughout a network as disclosed and claimed in two patent applications filed by the  
17 assignee of the present invention: "Managing a Distributed Directory Database",  
18 Krishnan et al, U.S. Serial No. 09/965430, filed September 27, 2001 and "Resolving  
19 Multiple Master Node Conflict in a DDB", Krishnan et al, U.S. Serial No. 09/964977,  
20 filed September 27, 2001, both of which are incorporated herein by reference in their  
21 respective entireties.

## 22 23 SUMMARY OF THE INVENTION

1           Embodiments of the present invention include apparatus, method, system, and/or  
2   computer program product for persisting or replicating data such as that in a datafile in a  
3   distributed computing environment. In one aspect of the present invention, a technique is  
4   provided for replicating a file in a computer network for use by a network user and  
5   having a plurality of nodes. A file is capable of being received from the user in any one  
6   of the nodes. In response to receiving the file in a certain one of the network nodes (the  
7   originator node), the file is replicated in all other nodes in the network. The file can be a  
8   new file or an updated file. The replication is performed in a manner that is network-  
9   topology independent and avoids a single point of failure.

10  
11           In another aspect of the present invention, another of the nodes is established as a  
12   master node and the plurality of nodes except for the originator and master nodes are  
13   slave nodes. The updated file is stored on the master node as a backup file. In each of  
14   the slave nodes, a particular file is updated corresponding to the updated file, provided  
15   that the particular file does not contain contents identical to contents of the updated file.  
16   Creation of the backup file is communicated (a success note) to the originator node and  
17   availability of the backup file is communicated to the slave nodes. The originator node,  
18   responsive to the success note, publishes a representation of the updated file to all other  
19   nodes (slave nodes and master node). Each of the slave nodes respond to the published  
20   representation by obtaining the updated file from the originator node. If the updated file  
21   is not obtained from the originator node, it is obtained from the backup file in the master  
22   node. The originator node establishes an updated file version variable as the  
23   representation which gets published to the slave nodes. A particular file version variable

1 corresponding to the particular file is established. Change from the particular file version  
2 variable to the updated file version variable is observed in each of the slave nodes, and  
3 responsive to each observation, the updated file is downloaded from the originator node  
4 into the particular file.

5  
6 In yet another aspect of the present invention, the updated file is received in local  
7 workspace in the originator node. Global workspace is operatively coupled to the local  
8 workspace in the originator node and receives the updated file from the local workspace  
9 in preparation to download it to any of the slave nodes upon request therefrom. Further,  
10 in the master node, there is master node local workspace for receiving the updated file  
11 whereupon its file version variable is error-checked to confirm validity of the file version  
12 variable. Also, master node global workspace is adapted to receive the updated file from  
13 the master node local workspace and the updated file is transferred to the global  
14 workspace if the validity of the file version variable is confirmed. Responsive to transfer  
15 of the updated file into the master global workspace the creation of the backup file is  
16 communicated to the originator node and the availability of the backup file is  
17 communicated to the slave nodes. If the validity of the file version variable is not  
18 confirmed by error-checking, an error is flagged, operation of the file replication  
19 procedure is stopped with respect to the file having the invalid file version variable, and  
20 the present file replication invention is prepared to receive a next successive updated file.

21  
22 However, the file replication procedure may continue with respect to other files  
23 previously received by the same originator which received the invalid file version

1 variable or previously received by other originators within the network. It should be  
2 understood that the updated file could be supplied by any one of multiple sources such as  
3 a human network user or other non-human software providers such as a security provider.  
4 It should also be understood that more than one new file can be received from these  
5 sources seriatim and replicated. (The terms "updated file" and "new file" can be used  
6 interchangeably herein, since an "updated" file contains "new" data compared to its prior  
7 file version; however, a "new file" can also refer to a file that had no prior existence.) It  
8 should be yet further understood that when the network is interacting with multiple users  
9 concurrently, each of whom selects a different network node to be its originator node,  
10 that multiple operations can take place simultaneously whereby a given node can  
11 simultaneously be both originator node with respect to a first user while being a slave  
12 node with respect to a second user. Thus, there can be multiple originator nodes at any  
13 given point in time, each originator node relating to or characterizing a different  
14 independent network scenario within the same network. However, the node selected as  
15 master node is master for all originator nodes and all slave nodes at any given point in  
16 time, and there can be only one master node at any given point in time.

17  
18 In yet another aspect of the present invention, to ensure replication if any of the  
19 foregoing operation fails to achieve satisfactory file replication, a first syncing thread is  
20 applied. Each of the slave nodes periodically polls the master node to determine if the  
21 slave node's particular file contents matches the updated file contents stored as backup in  
22 the master node. If there is no match, the particular and updated file contents are  
23 synchronized. A Directory Provider Service (DPS) version number is established to



1 identify the current version of the DPS in the network. A Directory Data Base (DDB)  
2 version number is established to identify the current version of the DDB in the network.  
3 The DPS version number on each of the slave nodes is compared with the DPS version  
4 number on the master node to obtain a respective DPS version number match. If there is  
5 a match for each of certain of the slave nodes, further synchronizing operation with  
6 respect to the current poll by each certain slave node is terminated. For the remainder of  
7 the slave nodes, where a DPS version number match was not obtained, a DDB version  
8 number match is achieved. In addition, for the remainder of the slave nodes, the  
9 particular file version variable on each of the remainder is compared with the updated file  
10 version variable on the master node to obtain a respective file version variable match. If  
11 a file version variable match is obtained for a portion of the remainder of the slave nodes,  
12 further synchronizing operation with respect to the current poll by each of the slave nodes  
13 in the portion of the remainder is terminated. And, for each slave node in the remaining  
14 portion of the remainder of slave nodes for which a file version variable match was not  
15 obtained, such a match is achieved. Thereby, the particular file contents matches the  
16 updated file contents in each of the slave nodes.

17  
18 As earlier noted, a particular node can be in the position of being a slave node  
19 with respect to multiple originator nodes, where each originator node relates to a different  
20 independent network scenario within the same network at the same time. Therefore, that  
21 particular slave node needs to ensure synchronization with respect to each new or  
22 updated file associated with each originator node. Accordingly, the foregoing polling is  
23 undertaken by each slave node of the sole master node with respect to each of that slave

1 node's originator nodes. The global administrator can configure the interval duration of  
2 polling operations for the entire network. The interval duration will be the same for each  
3 node in the network, but each node's interval may be offset from the other nodes'  
4 intervals. In other words, each node's interval may have a different starting time, but  
5 must have the same duration. A typical interval duration may be configured by the global  
6 administrator to be approximately five minutes.

7  
8 In yet another aspect of the present invention, each of the plurality of nodes is a  
9 storage system having storage media such as storage disk(s) on which both the particular  
10 file version variable and the particular file contents are stored and a second syncing  
11 thread is applied. The particular file version variable stored on the media in each of the  
12 nodes is compared, through periodic polling, with the particular file version variable  
13 stored elsewhere in the respective node to determine a particular file version variable  
14 match for each of the nodes. For certain of the nodes for which such match was not  
15 achieved, the two file version variables are synchronized and the particular file version  
16 variable match is achieved for each of the certain nodes. Synchronization is  
17 accomplished differently as a function of the cause of mismatch. The mismatch can  
18 result from extra files on disks or missing files on disks as compared to files on the node.  
19 Synchronization is also accomplished differently for master node file version variable  
20 mismatch as compared with other nodes. This additional periodic polling of the second  
21 syncing thread can take place at five minute intervals, but is not restricted to any  
22 particular interval. This additional periodic polling need not be coincident with the  
23 FRS/DDB polling of the first syncing thread described earlier.

1  
2           It is thus advantageous to utilize embodiments of the present invention in  
3 computer networks, such as computer storage networks, where data replication  
4 throughout the network is necessary or desirable. Accordingly, such data replication is  
5 achieved in a manner that avoids configuring the network in accordance with a particular  
6 topology, and in a manner to avoid a single point of failure since backup is provided as  
7 well as syncing threads.

8  
9           It is thus a general object of the present invention to provide an improved  
10 computer network.

11  
12           It is another general object of the present invention to provide an improved data  
13 replication facility for distributed computing environments.

14  
15           It is yet another object of the present invention to provide a technique for  
16 replicating data files in a network that is topology-independent and avoids a single point  
17 of failure.

18  
19           It is still another object of the present invention to provide an improved computer  
20 data storage network which utilizes a distributed File Replication Service based on at  
21 least a distributed Directory Provider Service.

1 Other objects and advantages will be understood after referring to the detailed  
2 description of the preferred embodiments and to the appended drawings wherein:  
3

#### 4 **BRIEF DESCRIPTION OF THE DRAWINGS**

5 Fig. 1 is a schematic block diagram of computer network of the type in which  
6 embodiments of the present invention can be particularly advantageous;

7 Fig. 2 is a more detailed block diagram of a computer network of the type shown  
8 in Fig. 1, showing FRS and DPS functionality in each network node;

9 Fig. 3 is a further detailed schematic block diagram of a node of the type  
10 displayed in Fig. 2 showing structure internal to FRS and DPS functionality and the  
11 various clients which network nodes of this type can serve;

12 Fig. 4 is a schematic block diagram showing a network's master, slave, and  
13 originator nodes and their interrelated functionality when operating to receive a new or  
14 updated file from a user client and to update the slave node's corresponding file in  
15 accordance with principles of the present invention;

16 Fig. 5 is a flowchart depicting a first portion of an algorithm implemented by  
17 operation of the present invention;

18 Fig. 6 is a flowchart connected from Fig. 5 depicting a second portion of the  
19 algorithm of Fig. 5 implemented by operation of the present invention;

20 Fig. 7 is a flowchart connected from Fig. 6 depicting a third and final portion of  
21 the algorithm of Fig. 6 implemented by operation of the present invention;

22 Fig. 8 is a flowchart depicting a first syncing thread of a recovery algorithm  
23 which is also utilized in Fig. 7; and,

Fig. 9 is a flowchart depicting a second syncing thread of a recovery algorithm which is also utilized in Fig. 7.

## DESCRIPTION OF THE PREFERRED EMBODIMENTS

### Figure 1 – Network Overview

Referring to Fig. 1, there is depicted a schematic block diagram of a computer network of the type in which embodiments of the present invention can be particularly advantageous. This network can represent a computer data storage network, or can represent another type of client-server network. Network 101, as a computer data storage network, includes master node 103 and slave nodes 104-110 which are all operatively and communicatively inter-coupled by way of bus structure 111. Each slave node is shown to be communicatively linked to a (human) user: slave node 104 is coupled to user 112 via bidirectional link 113; node 105 is coupled to user 114 via link 115; node 106 is coupled to user 116 via link 117; node 107 is coupled to user 118 via link 119; node 108 is coupled to user 120 via link 121; node 109 is coupled to user 122 via link 123; and node 110 is coupled to user 124 via link 125. Node 109 is also shown communicatively linked to user 120 via link 121A while also being coupled to user 122, more discussion of which shall be presented hereinbelow. In the upper left hand corner of the diagram is global administrator 102, a user having special privileges as compared with the other users shown. The global administrator is communicatively coupled to every node in the network, including the master node, but these connections are not shown for purposes of enhancing clarity of presentation. The global administrator is the only user who can

1 establish or change the master node, and is further described in the incorporated by  
2 reference patent applications.

3  
4 Nodes 104-110 are labeled “slave” nodes to distinguish them from “master” node  
5 103 (master and slave nodes are defined in the incorporated by reference patent  
6 applications). However, since each slave node is shown in communication with at least  
7 one user, then each slave node can be in receipt of a new data file or updated data file  
8 forwarded to it by its respective user (in the vernacular: “the new data file is dropped on  
9 the node”). Any slave node receiving a new or updated data file from its user is re-  
10 defined herein as an “originator node” with respect to that new or updated data file being  
11 dropped on it. In a first scenario, for example, if user 114, by way of a graphical user  
12 interface (GUI – not shown) through which the user communicates with node 105, drops  
13 a new file on node 105, then node 105 becomes the originator node with respect to that  
14 particular new file for the rest of the network. In this scenario, the network node status  
15 is: master node 103 retains its master status, and slave nodes 104 and 106-110 each  
16 retains its respective slave status with respect to that new particular file. This network  
17 node status would also apply to any successive new files forwarded to node 105 by user  
18 114. However, in a second scenario, a different network node status would result if a  
19 different user, for example user 116, were to drop a different new file on slave node 106.  
20 In this second scenario, slave node 106 then becomes originator node for this different  
21 new file with respect to the same master node 103, and nodes 104, 105, and 107-110 are  
22 slave nodes with respect to originator node 106. These two different scenarios can occur  
23 simultaneously, where a particular slave node can be both a slave node with respect to a

1 particular data file and an originator node with respect to a different data file at the same  
2 time. The number of these different scenarios can grow with the total number of nodes  
3 and users involved in the network, where all new data file operations from all users on all  
4 slave nodes can occur simultaneously in the manner just described.

5  
6 User 120 is communicatively coupled to both slave node 108 via link 121 and  
7 slave node 109 via link 121A. This illustrates the fact that a user can be concurrently  
8 coupled to more than one node. As a network of storage systems, this shows a user being  
9 coupled to more than one storage system at the same time. In this case, storage system  
10 108 becomes originator node for the network with respect to a new data file supplied to it  
11 by user 120, while storage system 109 becomes originator node for the network with  
12 respect to a different new data file supplied to it by the same user 120. Further detail  
13 regarding construction and operation of this network, by which replication (duplication)  
14 of these new data files is achieved throughout the network, is presented hereinbelow in  
15 connection with other Figures.

### 16 17 **Figures 2 & 3 – Operational Overview**

18 Figure 2 may be a more detailed block diagram of a portion of the network of Fig.

19 1. Therefore, network 201 may correspond to a portion of network 101; master node 203  
20 may correspond to master node 103; user/GUI 214 may correspond to user 114;  
21 originator node 205 may correspond to originator/slave node 105; slave nodes 204 and  
22 207 may correspond to slave nodes 104 and 107 respectively; global administrator 202  
23 may correspond to network administrator 102; interconnecting link 215 may correspond

1 to link 115; and inter-nodal direct links 210, 206, 208, 216, and 209 may collectively  
2 correspond to a portion of bus 111.

3  
4 A File Replication Service (FRS) and a Directory Provider Service (DPS) are  
5 depicted as being distributed throughout network 201 and reside in each node of network  
6 201 as FRS 211 and DPS 212. (Other services and functionality included within these  
7 nodes, for example, storage processors and disk drives for nodes which comprise a  
8 storage system, are not shown in order to enhance clarity of presentation.) FRS 211 is  
9 dependent upon or runs on DPS 201 and the contiguous juxtaposition of blocks  
10 representing these two services is intended to suggest this operational relationship.  
11 Within each DPS 212 is depicted a directory data base, DDB 213, which has been  
12 described in detail in the incorporated by reference patent applications.

13  
14 By way of an introductory overview of network operation, one may assume that  
15 global administrator 202 had selected node 203 to be the network's master node.  
16 Selection of the master node, and communication of the selection to network nodes, are  
17 described in the incorporated by reference patent applications. User 214, through its GUI  
18 which is similar to the graphical user interface described in the incorporated by reference  
19 patent applications, "drops" a new file on slave node 205 thus making it the originator  
20 node for user 214 in network 201. Originator node 205, through its FRS 211  
21 communicates with FRS 211 in master node 203 over link 206, to advise the master node  
22 of this newly arrived data file from user 214. If certain characteristics of this new data  
23 file are acceptable to master node 203 (detailed below) it will retain a copy of the



1 contents of such file for backup purposes. The retention of the file for these purposes is  
2 communicated back to FRS 211 in originator node 205 which then causes its DPS 212 to  
3 “publish” a representation of the new file to all slave nodes in the network, namely nodes  
4 204 and 207 in this example. In other words, DPS 212 in originator node 205 sends a  
5 new file representation (termed a “file version variable” or FVV) to all slave nodes in the  
6 network. In this instance the FVV is sent to both DPS 212 in slave node 204 over link  
7 216 and DPS 212 in slave node 207 over link 209, advising both slave nodes of a new  
8 data file being introduced into the network. Through a manipulation within each slave  
9 node to be discussed below, FRS 211 of slave node 204 and FRS 211 of slave node 207  
10 each obtains the new data file from FRS 211 of originator node 205 over links 216 and  
11 209 respectively. If not obtainable from the originator node by either or both slave  
12 nodes, the new data file is obtained from the backup copy retained in master node 203  
13 over links 210 and 208 respectively. In addition, there is periodic synchronizing  
14 (syncing) undertaken by each slave node with the master node over links 210 and 208  
15 which brings any data files up-to-date that might have remained outdated for any reason,  
16 even if the slave nodes had sought a new data file from the master node’s backup copy.

17  
18 Referring next to Fig. 3, there is provided a further detailed schematic block  
19 diagram of a node of the type displayed in Fig. 2. It shows functionality internal to FRS  
20 and DPS services used for manipulating data files and their representations, and shows  
21 various clients served by network nodes of this type. FRS 303 and DPS 302 are shown  
22 within node 301 and are operatively intercoupled by bidirectional link 317.

1 • FRS 303 includes data file 304, local workspace 306 and global workspace 307  
2 functionality, some of which may be used by the node when functioning as a slave  
3 node and other of which may be used by the node when functioning as originator or  
4 master node. When functioning as originator or master node, local workspace 306  
5 and global workspace 307 are operational. When functioning as slave node, data file  
6 304, including its data structure (FVV 305 and data 305D), are operational.

7 • DPS 302 includes file name 312, file IP address 311, FVV 310, observer object 313,  
8 and FVV publish buffer 318 functionality, some of which may be used by the node  
9 when functioning as a slave node and other of which may be used by the node when  
10 functioning as originator or master node. When functioning as originator node,  
11 publish buffer 318 is operational. When functioning as slave node, file name 312, file  
12 IP address 311, FVV 310, and observer object 313 are operational.

13 An observer is a software object which is fixated on another object, and it has the  
14 capability of triggering, or transmitting a signal, simultaneously with any change in that  
15 other object. In various embodiments of the present invention, an observer can fixate  
16 onto an object which holds an old file version variable which changes to an updated file  
17 version variable when an old file is updated, and the same or another observer can fixate  
18 onto another object which reflects a new file version variable upon occurrence of a new  
19 file being added to the network. “Updated file version variables” and “new file version  
20 variables” are terms that may be used interchangeably herein. It should be understood  
21 that other functionality (not shown) may be included in both FRS 303 and DPS 302.

22

1           When node 301 is operating as an originator node, local workspace 306 receives  
2   data structure(s) 316 which has an FVV header and a data load portion as shown. The  
3   data structure(s) can be supplied by a number of suppliers: GUI user 308 as discussed,  
4   security provider software 314, and/or other software provider and/or users 315. Local  
5   workspace 306 holds the FVV and its associated data load as an input buffer. Part of the  
6   data structure in the buffer is first manipulated and tested before it is permitted to feed  
7   global workspace 307 as shown . Detail regarding this manipulation and testing, which  
8   involves the master node, is provided in connection with Fig. 4. Once transferred to  
9   global workspace 307, however, the data structure is then available for downloading to  
10   slave nodes in a manner that provides the actual data file replication to the slave nodes.  
11   DPS 302 is readied by FRS 303 via link 317 to broadcast or publish (as the originator  
12   node) from FVV publish buffer 318 the FVV portion of the updated data structure as well  
13   as the updated file's name and IP address to all of the slave nodes. The slave nodes  
14   manipulate the published FVV, described below, after which the originator node's global  
15   workspace 307 is accessed and read by the slave nodes, whereby downloading of the data  
16   load takes place resulting in actual data file replication to the slave nodes.

17  
18           By contrast, when node 301 is operating as a slave node, the local and global  
19   workspaces are not used in FRS 303 and FVV publish buffer 318 is not used in DPS 302.  
20   Rather, when operating as a slave node, a prior version of a data structure such as  
21   representation FVV 305 and its corresponding data 305D comprise data file 304 stored in  
22   FRS 303. Any update to that data structure is observed by observer 313 while fixated  
23   upon the FVV 305 object which becomes file version variable FVV 310, in response to

1 the originator node's publishing that updated file version variable along with its file name  
2 and IP address as described above. This causes slave node 301 to make a comparison  
3 between FVV 310 and FVV 305 which do not match. The mismatch results in slave  
4 node 301 seeking the updated file with file name 312 at IP address 311 that corresponds  
5 to FVV 310. The updated file is first sought from the originator node, and, failing that is  
6 next sought from the master node which is holding the updated file as backup. This  
7 overview discussion of operation is detailed further in connection with discussion of Figs.  
8 4-9 hereinbelow.

#### Figure 4 – Detailed Description

Fig. 4 is a schematic block diagram showing a network's master, slave, and  
originator nodes and their interrelated functionality when cooperating to receive a new  
file from a user client and update the slave node's corresponding file in accordance with  
principles of the present invention. In the upper left hand section of the diagram is  
master node 420 including its DPS 421 and FRS 422 which, in turn, includes global  
workspace 423 and local workspace 424. In the upper right hand section of the diagram  
is originator node 430 including its DPS 431 which, in turn, includes FVV publish buffer  
435. Originator node 430 further includes its FRS 432 which, in turn, includes global  
workspace 433 and local workspace 434. In the lower left hand section of the diagram is  
slave node 440 including its DPS 441 which, in turn, includes observer object 442 and  
FVV 443. Slave node 440 further includes its FRS 444 which, in turn, includes data file  
445 with its corresponding FVV 446 and data 446D. In the lower right hand section of  
the diagram disk storage component 460 and GUI user 450 are shown. These nodes,

1 components and interfaces are operatively coupled as shown and to be described in detail  
2 hereinbelow.

3  
4 As noted earlier, these nodes can be physically very similar to each other, if not  
5 identical. However, in Fig. 4, only functionality which is operative in a node is shown in  
6 that node. Functionality which is not operative in that node is intentionally not shown in  
7 that node, to enhance clarity of presentation. Furthermore, and as earlier noted, there can  
8 be yet additional functionality included in any particular one or more of these nodes  
9 which may be essential or desirable for overall functioning of that node within a storage  
10 network or other operational environment. However, such additional functionality would  
11 be irrelevant to operation of embodiments of the present invention and therefore is not  
12 shown herein.

13  
14 In operation, GUI user 450 decides to update an existing data file in a network of  
15 the type shown in Fig. 1 and, in accordance with our nomenclature, drops a new or  
16 updated file on local workspace 434 in originator node 430 via external communication  
17 link 401. External communication links conform to Transmission Control  
18 Protocol/Internet Protocol (TCP/IP), the suite of communication protocols used to  
19 connect computer systems on the Internet. In a preferred embodiment, these links can  
20 conform to the Ethernet LAN architecture, or versions thereof. This new file is identical  
21 in format to that which contained data structure 316 depicted in Fig. 3, having both a  
22 FVV and a data load. Local workspace 434 is similar to a "scratch pad" memory or a  
23 "clipboard" memory and serves as a temporary storage for the new file while other

important functions are undertaken before arranging for the new file to be more permanently stored. Local workspace 434 transmits the new file via network communication link 402 (the network communication link is a compatible Ethernet bus) to local workspace 424 (which may be similar or identical in functionality to local workspace 434) located in master node 420. This transmission takes the form of an XML language request, which is a textual way of representing objects. On the receiving end of this transmission in the master node, a parser (not shown) of this XML request, which determines what sort of object was received, has special characters used for calibration. Since a binary file could have those same special characters in it, the binary file could be problematic for an XML parser which could mistake those embedded special characters as calibration characters. Therefore, to transmit a binary file, a conversion from binary to XML is needed and 64 bit encoding can be used. The encoded binary file is wrapped inside XML code so that the master node's parser receiving the encoded file into local workspace 424 can understand what it is receiving. The lines of code for a typical encoded binary file wrapped in XML for handling this transmission can be similar to that shown in Table I.

---

**TABLE I – NEW DATA FILE AS XML-WRAPPED BINARY CODE**

---

```
<?xml version="1.0" ?>
<CIM CIMVERSION="2.0" DTDVERSION="2.0" >
<MESSAGE ID="877" PROTOCOLVERSION="1.0" >
<SIMPLEREQ>
<METHODCALL NAME="UploadFile" >
```

1 <LOCALNAMESPACEPATH><NAMESPACE NAME="root" /><NAMESPACE  
2 NAME="e" />  
3 <NAMESPACE NAME="navisphere" /></LOCALNAMESPACEPATH>  
4 <PARAMVALUE NAME="InstanceName" >  
5 <INSTANCEPATH>.<NAMESPACEPATH>  
6 <HOST 127.0.0.1/>.<LOCALNAMESPACEPATH>  
7 <NAMESPACE root/>.<NAMESPACE e/>.<NAMESPACE n/>  
8 </LOCALNAMESPACEPATH>.</NAMESPACEPATH>.<INSTANCENAME  
9 CLASSNAME="AAA">.  
10 <KEYBINDING NAME="uniqueKey"><KEYVALUE  
11 VALUETYPE="string">BBB</KEYVALUE>  
12 </KEYBINDING>.</INSTANCENAME>.</INSTANCEPATH>.</PARAMVALUE>  
13 <PARAMVALUE NAME="FileName" Type="string">  
14 <VALUE>notes.txt</VALUE></PARAMVALUE>  
15 <PARAMVALUE NAME="Buffer"  
16 Type="string"><VALUE>ABCDEF</VALUE>.</PARAMVALUE>  
17 <PARAMVALUE NAME="BufferSize"  
18 Type="string"><VALUE>6</VALUE>.</PARAMVALUE>  
19 <PARAMVALUE NAME="NeedToRegister"  
20 Type="boolean"><VALUE>true</VALUE>.</PARAMVALUE>  
21 </METHODCALL>.</SIMPLEREQ>.</MESSAGE>.</CIM>  
22  
23

---

1  
2       Upon receipt of this new file in its local workspace, master node 420 first  
3 performs an error checking operation on the FVV portion of the new file. In this  
4 operation, the master node determines if the new file's FVV is valid. In other words, the  
5 master node queries: does this FVV make sense compared with other FVV numbers with  
6 which it has familiarity? Or, is this FVV garbled or otherwise unusable? The master  
7 node needs to assure itself that it can retrieve this new data file based on this FVV if it is  
8 required to do so, and needs to examine this FVV for this purpose in advance of  
9 permanently accepting the data file. (Since the user can save any type of data in this file  
10 the master node cannot verify substantive data for correctness – it can only examine the  
11 header FVV number for “reasonableness” relative to other FVV numbers.) If the FVV  
12 does not pass the error check, the operation stops, the master node flushes the new file  
13 from its local workspace, and an error flag is returned to the user's GUI. The user can try  
14 again later or try to use a different node in the network as its originator node. However,  
15 if the FVV does pass the error check, local workspace 424 transfers the new data file to  
16 global workspace 423 via intra-nodal communication link 403. Intra-nodal  
17 communication links are Ethernet compatible. Global workspace 423 is a permanent area  
18 of segregated memory located within the storage system (not shown) associated with  
19 master node 420.

20  
21       Immediately upon permanently storing the new data file in global workspace 423,  
22 an acknowledgment, termed a “success note” is returned by master node 420 to FRS 432  
23 within originator node 430 via network communication link 404. The success note is



1 also in XML and is encoded similarly to that shown in Table I. The receipt of this  
2 success note by FRS 432 is its signal that master node 420 has indeed permanently  
3 accepted the new data file and that FRS 432 should also permanently accept the new data  
4 file. FRS 432 thereupon allows its local workspace 434 to transfer the new data file  
5 temporarily stored therein to its global workspace 433 over intra-nodal communication  
6 link 405. Global workspace 433 is a permanent area of segregated memory located  
7 within the storage system (not shown) associated with originator node 430. The new data  
8 file is stored in global workspace 433 for purposes of making it available to be read by  
9 and downloaded to the slave nodes of the network. But, before the downloading can take  
10 place, existence of the new data file must first be published or broadcast to the network's  
11 slave nodes as follows.

12  
13 FRS 432 "registers" the new data file with its DPS 431 by transferring the FVV,  
14 file name, and file IP address of the new data file into FVV publish buffer 435 located  
15 within DPS 431. This is accomplished over intra-nodal link 406. FVV publish buffer  
16 435 publishes or broadcasts the FVV value or number as well as corresponding file name  
17 and file IP address to all other network nodes via network communication links 407.  
18 This publication uses the network's directory provider service where each node,  
19 including this originator node, has a complete list of IP addresses of all other network  
20 nodes (see incorporated by reference patent applications). This publication or broadcast  
21 is again performed in XML.

1 In a typical network, this broadcast could comprise hundreds of separate  
2 communications to hundreds of nodes respectively, but in this simplified example only  
3 one slave node and one master node are shown in receipt of the published FVV number  
4 (and name and address). The FVV number is a Unique IDentifier (UID). Each FVV is a  
5 different number from any of the DDB version numbers of the incorporated by reference  
6 patent applications. The FVV numbers and the DDB version numbers are also different  
7 categories of numbers. Receipt by master node 420 of the published FVV is not needed  
8 by the master node to facilitate the file replication process, and is therefore a redundancy.  
9 In view of the fact that only one node out of typically hundreds of nodes can be master  
10 node at a given time, this singular redundancy per new data file is a small price to pay for  
11 savings in cost, effort and time that results from design simplification because of  
12 allowing the redundancy. However, although the master node does not need it, slave  
13 node 440 does need the published FVV to enable the replication process to proceed as  
14 follows.

15  
16 DPS 441 of slave node 440 receives the published FVV UID as FVV 443 (and  
17 corresponding file name and IP address as well, but not shown in this Fig.). Object  
18 observer 442 was observing or was fixated upon a different FVV UID which comprised  
19 the object being observed prior to arrival of FVV 443. The different FVV UID which  
20 was being observed by observer 442 represents the same data file as the one being  
21 updated by user 450, but represents that data file prior to being updated to the new data  
22 file by user 450. The prior data file is data file 445 represented by FVV 446 and  
23 containing data 446D, and is located in FRS 444 of slave node 440. Immediately upon

1 observing the change from a UID corresponding to FVV 446 to a different UID  
2 corresponding to FVV 443, observer 409 signals this change to FRS 444 via intra-nodal  
3 communication link 409. FRS 444 then immediately makes a comparison between the  
4 two FVV UIDs, FVV 443 versus FVV 446, and notes that the two UIDs are different.  
5 Slave node 440, operating through its FRS 444, then sends an XML request to the node  
6 (an originator node) which has the IP address corresponding to FVV 443 for a file with a  
7 name corresponding to FVV 443, the IP address and file name having been earlier  
8 published to slave node 440. This request is thus sent over network communication link  
9 410 to FRS 432 located within originator node 430 for the updated file corresponding to  
10 FVV 443. This request is received in global workspace 433 of FRS 432. Global  
11 workspace 433 is storing the updated file pursuant to its receipt of such file from local  
12 workspace 434 as earlier-described. If this file replication process is operating properly,  
13 in response to the XML request for the new file, global workspace 433 allows the new  
14 file to be read and downloaded to FRS 444 in slave node 440, thereby replacing old data  
15 446D with new data and replacing old FVV 446 with new FVV 443. For the situation  
16 where a new file is introduced into the network for the first time, the observer (or  
17 alternatively a different observer) notes the change from no file version variable to some  
18 file version variable and the operation is otherwise identical to that discussed above.

19  
20 With respect to the topic of proper operation of the file replication process, there  
21 are a variety of possible failure modes including but not limited to the following  
22 examples:

- 1     • One mode may involve a network node (master, originator or slave) which might  
2       become inoperable at any time. A network cord to that node could become  
3       unplugged if, for example, someone tripped over it, etc. This could cause internal  
4       failure in that node. Depending on which node failed and the precise time of its  
5       failure relative to the time of dropping a new file on an originator node, this failure  
6       mode might result in return of an error flag to the GUI user. If the master node fails  
7       or is otherwise unavailable, no new or updated files can be dropped by any user until  
8       the master node is made operative or until another node is promoted to master status  
9       as a substitute for the original master node.
- 10    • Another mode involves an unreachable master node. In this instance, when the  
11      originator node “pushes” the new file to the master node but where the backup  
12      operation is not completed, the originator node's file replication service is shut off  
13      until the master becomes reachable (or until another master is selected).
- 14    • Another mode relates to the master node's error checking on the FVV which results  
15      in a bogus or unreadable FVV. An error flag is returned to GUI as earlier described.
- 16    • Another mode involves a failed registration of the new file's FVV, file name, and IP  
17      address by the originator node's FRS into its DPS's FVV publish buffer. This  
18      disallows publishing of the new FVV to any of the slave nodes in the network.
- 19    • Another mode is that the new file's FVV has been published by the originator node  
20      but the originator node cannot be reached by any of the slave nodes to download the  
21      new data file because of a failed network link to the originator node, or a failed  
22      originator node itself.

1 These examples and virtually any other foreseeable failure mode can be handled by the  
2 backup master node and/or the two syncing threads.

3  
4 When the master node is called upon to serve in a backup capacity, slave node  
5 440 attempts to obtain the new file from the master node. The new file was stored as  
6 backup on global workspace 423 in master node 420, as earlier described. If that storage  
7 was not accomplished, the replication process would not have advanced beyond that  
8 point. Slave node 440 knows the IP address of its master node as a result of normal  
9 operation of its DPS 441 as described in the incorporated by reference patent  
10 applications. Slave node 440 now utilizes its FRS 444 with respect to the master node in  
11 a manner similar to how it was used in attempting to download the new file from the  
12 originator node, to obtain the new file by way of network communication link 411 from  
13 global workspace 423 in the master node. If this attempt by slave node 440 to obtain the  
14 new data file from master node 420 fails for any reason, there are two syncing threads  
15 that are periodically applied to the slave node, at approximately five minute intervals but  
16 not necessarily coincidentally or at the same frequency. These syncing threads are  
17 applied whether or not handling of a failure mode is needed. This ensures that all data  
18 files are properly updated for all slave nodes within a time period not to exceed the sum  
19 of those two intervals, or approximately ten minutes in this example. These syncing  
20 threads, one of which involves disk(s) 460 via network communication link 461, shall be  
21 described in more detail in connection with Figures 8 and 9 hereinbelow.

One possible alternative embodiment of the present invention is to have each of the slave nodes seek the updated file from the master node instead of from its originator node, but this embodiment may have drawbacks. First of all, since the master node is acting as backup then such backup protection may be foreclosed if the master node is also acting as the primary source of the new file. Secondly, a network load imbalance may result. If a number of originator nodes are serving a number of users, and each user drops new files on its originator node at approximately the same time, each slave node in the network shall attempt to download all of those new or updated files from only the master node. The master node's capacity to broadcast each new FVV and/or to permit downloading of each new data file from its global workspace may be overwhelmed. A load balancing effect is achieved with the earlier-described embodiment wherein each originator node, upon which a new file has been dropped, serves as the primary source of that new file for the network's slave nodes. This spreads the broadcasting and downloading over all originator nodes thereby achieving load balancing, rather than forcing all those operations upon a single master node and creating a load imbalance. In a similar vein, the master node can also be made the originator node by a user, but such an arrangement eliminates the useful backup service otherwise provided by the master node. However, other alternative embodiments which are advantageous are discussed hereinbelow.

### **Figures 5, 6, & 7 – Interconnected Flowcharts**

Figure 5 is a flowchart depicting a first portion of an algorithm implemented by operation of the present invention. In block 501 a user, such as any user of Fig. 1 or GUI

1 user 450 of Fig. 4, drops a new file on a network node, thus making that node an  
2 originator node. In Fig. 4, node 430 becomes the originator node when the new file is  
3 received in local workspace 434. The algorithmic process moves to block 502 wherein  
4 the originator node's local workspace sends the new file to the master node's local  
5 workspace, as, for example, receipt of the new file into local workspace 424. The  
6 algorithmic process proceeds to decision block 503 wherein the query is made: does the  
7 file version variable representing or identifying the data in the new data file pass an error  
8 check performed by the master node? On the one hand, if "no", the algorithmic process  
9 moves to block 504 where both of the local workspaces in the originator and master  
10 nodes are purged, an error flag is returned to the user's GUI, and the algorithmic process  
11 stops. On the other hand, if "yes", the algorithmic process moves to block 505 wherein  
12 the master node transfers the new data file from its local workspace to its global  
13 workspace (423 in Fig 4) for subsequent backup support to a slave node if needed. The  
14 algorithmic process moves next to block 506 where a "success" message is returned by  
15 the file replication service in the master node to the file replication service in the  
16 originator node, thereby advising the originator node that the new file's FVV has passed  
17 the master node's error check and that the master node has stored the new file including  
18 at least its name, FVV and data load into its more-permanent global workspace. In block  
19 507, only after the originator node receives the success note does it allow transfer of the  
20 new file from its local workspace to its global workspace (433 in Fig. 4) in preparation  
21 for eventual "pulling" from, or reading by, the network's slave nodes. The process  
22 continues to the "A" tab in Fig. 6.

1 In block 601 of Fig. 6, the originator node's file replication service registers the  
2 file version variable and file name with its directory provider service which includes the  
3 originator node's IP address in the registration. (As detailed in the incorporated by  
4 reference patent applications, part of the responsibility of the directory provider service is  
5 to keep track of all network node IP addresses including its own address.) This  
6 registration is retained in a buffer such as FVV publish buffer 435 in Fig. 4. In the next  
7 step of the algorithm in block 602, the originator node's directory provider service  
8 broadcasts or publishes the registered file version variable along with the file name and  
9 originator node's IP address to the entire network including master and all slave nodes.  
10 The process moves next to decision block 603 wherein the query is made: for each slave  
11 node, did it receive the published file version variable? On the one hand, if "no", then for  
12 each node for which the published file was not received (non-updated nodes) the process  
13 moves to tab "B" in Fig. 7. These non-updated nodes remain in that state until syncing  
14 threads in special recovery algorithms are applied to them in steps 703 and 704 of Fig. 7.  
15 These recovery algorithms are detailed in Figs. 8 and 9 and are discussed hereinbelow.  
16 But, on the other hand, if "yes", then for each slave node that did receive the published  
17 file the algorithmic process moves to block 604. In block 604 an object observer in the  
18 slave node's directory provider service is fixated upon the file version variable associated  
19 with the old data file corresponding to the updated data file prior to its being updated by  
20 the user. The object observer notes a change in the file version variable which is "notice"  
21 to the slave node's directory provider service that the file itself (data load) has changed or  
22 has been updated. (The same result obtains if the user introduces a data file that had no  
23 prior incarnation as contrasted with the user "updating" an old data file because in the



1 former case there is an observed FVV change from no FVV value to a finite FVV value.  
2 In an alternative embodiment, to handle new file additions, a separate “new file” object  
3 observer may be used, obtaining the same result.) The algorithmic process then moves to  
4 block 605 wherein the slave node’s observer signals the change to its own file replication  
5 service to allow it to obtain the new file. This corresponds to observer 442 observing  
6 FVV 443 as an updated file version variable received by way of publication from  
7 originator 430, and sending the change signal over link 409 in Fig. 4 to FRS 444. The  
8 algorithmic process then moves to block 606 wherein the slave node’s file replication  
9 service reads or pulls the new data file from the global workspace of the originator node,  
10 the IP address of which is known to the file replication service since it was received along  
11 with the updated file version variable and file name in the originator’s publication  
12 thereof. The process moves to tab “C” in Fig. 7.

13  
14 In Fig. 7, decision block 701 presents the query: for each slave node, is the new  
15 file successfully pulled from the global workstation of the originator node? On the one  
16 hand, if “no”, the process moves to block 702 wherein for each slave node for which the  
17 new file was not successfully read from the global workspace of the originator node, the  
18 new file is obtained from the global workspace of the master node. In a network of  
19 multiple slave nodes, most may have operative links to the originator node but a few may  
20 not, whereby the new file might not be obtainable by all network nodes from the  
21 originator node. Since the master node contains a backup copy of the updated file, it can  
22 be obtained by a slave node that could not obtain it from the originator node. But, it is  
23 possible that the backup operation in block 702 also might fail for certain nodes under

1 certain conditions – again, a bad link from slave to master node may be a problem.  
2 However, any such node failing in its operation of block 702, and which therefore failed  
3 twice in its attempt to get an updated file, first from its originator node and then from its  
4 master node, is given yet another opportunity to obtain this updated file. This  
5 opportunity is presented by way of operations specified in blocks 703 and 704 to be  
6 discussed further hereinbelow. However, before such discussion is undertaken, consider  
7 the other result from query block 701. On the other hand, if “yes”, then the process  
8 moves from block 701 directly to blocks 703 and 704. Thus, regardless of success or  
9 failure by any particular slave node in its attempt to read a new file from its originator  
10 node, the algorithmic process converges at this point for all nodes. In block 703 a first  
11 syncing thread is applied to all nodes in the network; and, in block 704, a second syncing  
12 thread is likewise applied to all nodes. These two syncing threads comprise a recovery  
13 algorithm which is repetitively and periodically applied to all nodes to further ensure  
14 synchronization (that all slave nodes have the latest network files if not obtained by other  
15 operation of the network’s file replication service), and are detailed in Figs 8 and 9 as  
16 follows.

### 18 **Figures 8 & 9 - Recovery Algorithm Flowcharts**

19 As previously described, there can be a number of reasons for a network to  
20 become unsynchronized whereby any one or more nodes in the network contains data  
21 files that are not current. A list of failure modes or scenarios was provided above which  
22 is by no means a complete list. And those failure modes need not occur in the alternative,  
23 but can occur together. Accordingly, despite effective operation of the file replication

1 service described thus far, by which files are replicated virtually instantaneously with  
2 their insertion into the network, to further ensure that every node in the network, does not  
3 run in an unsynchronized mode for longer than a short predetermined interval because of  
4 various permutations and combinations of failure scenarios that might occur, syncing  
5 threads of Figs. 8 and 9 are included as part of the file replication service. In Fig. 8, a  
6 synchronization is obtained between the directory provider service and the file replication  
7 service for each node in the network. In Fig. 9 a synchronization is obtained between the  
8 directory provider service (acting through the file replication service) and data stored on  
9 the storage disk(s) for each node in the network. These two syncing threads, in a  
10 preferred embodiment, repetitively operate at approximately five minute intervals.  
11 However, the two threads need not operate simultaneously – they can be offset from each  
12 other, need not run at the same or similar intervals, and can be set to run at shorter or  
13 longer intervals than five minutes.

14  
15 In the incorporated by reference patent applications, a polling feature was  
16 discussed in detail with regard to version numbers associated with only the directory  
17 database (DDB) portion of the directory provider service. However, the directory  
18 provider service handles two kinds of data: IP address data associated with DDB and  
19 FVV data associated with data files, but not the data loads per se identified by those FVV  
20 numbers. Accordingly, there are other version numbers associated with the directory  
21 provider service, DPS version numbers, which take into account both the DDB and the  
22 FVV. Therefore, the polling feature described in the incorporated by reference patent

1 applications, which is limited to IP address data, operates independently of the syncing  
2 thread (polling) of Fig. 8.

3  
4 In Fig. 8, with reference to block 801, for each slave node there is periodic polling  
5 by the slave node's directory provider service of its master node's directory provider  
6 service to determine if the slave node's directory provider service version number  
7 matches the master node's directory provider service version number. (In Fig. 8, each  
8 slave node is intended to mean "participating node" in the network or configuration  
9 which is as defined in the incorporated by reference patent applications as any node in the  
10 network or configuration other than the master node.) The algorithmic process moves  
11 from block 801 to query block 802 where the query is made: do the master and slave  
12 directory provider service version numbers match? If "yes", the algorithmic process  
13 stops because the match indicates that both the DDB version numbers match properly  
14 (implying that the network is fully apprised of the whereabouts of all of its network  
15 nodes) and the FVV numbers match properly (implying that the all of the network's data  
16 files are up to date). But, if "no", then the algorithmic process continues in decision  
17 block 803 where the query is posed: is the mismatch in the directory provider service  
18 version numbers due to DDB version number mismatch only?

19  
20 On the one hand, if the answer to the query in decision block 803 is "yes", the  
21 algorithmic process moves to block 806 wherein IP address replication occurs in each  
22 network node in accordance with the incorporated by reference patent applications, after  
23 which the algorithmic process stops. On the other hand, if the answer to the query in

1 block 803 is “no”, the algorithmic process moves to another decision block, block 804,  
2 wherein another query is posed: is the mismatch in the directory provider service version  
3 numbers due to the file replication service’s FVV mismatch only?  
4

5 On the one hand, if the answer to the query in decision block 804 is “yes”, the  
6 algorithmic process moves to block 807 wherein for each mismatched file version  
7 variable on that slave node, the slave node’s file replication service obtains a new file  
8 version variable and its associated new file from the global workspace in the file  
9 replication service of the master node. Again, the master node is serving in its backup  
10 capacity relative to the file replication service, after which the algorithmic process stops.  
11 On the other hand, if the answer is “no” this means that the mismatch must be due to a  
12 combination of DDB version number mismatch and file version variable mismatch. The  
13 algorithmic process moves to block 805 wherein the actions of both blocks 806 and 807  
14 are performed as a combined solution to the combined mismatch, after which the  
15 algorithmic process stops.  
16

17 In Fig. 9, with reference to block 900, for each node in the network including  
18 master node, originator node and all slave nodes, through its respective file replication  
19 service, the contents stored on the node’s physical storage disk(s) are periodically polled  
20 by comparing each disk-stored file version variable with its respective file version  
21 variable stored in the node’s directory provider service. This comparison is made  
22 periodically for each file version variable on each disk of each node in the network, the  
23 period being typically about five minutes between polls. In Fig. 4, storage disk(s) 460 is

1 shown operatively coupled to slave node 440. For example, FVV 443 in DPS 441 is  
2 compared with its corresponding FVV in Disk 460. The other nodes in that figure also  
3 have storage disk(s) operatively coupled thereto, but are not shown for purposes of  
4 enhancing clarity of presentation. It should be understood that other structure and  
5 functionality may be necessary for the network of Fig. 4 to function as a storage network,  
6 such as storage processors, etc., which are also not shown for the same reasons. The  
7 algorithmic process moves next to decision block 901 wherein the query is posed: for  
8 each comparison is there a file version variable match? If "yes" the data files associated  
9 with those matching file version variables also match and the algorithmic process  
10 concludes for that particular file version variable on that particular disk for that particular  
11 node in the network. But, if "no", then the algorithmic process moves to decision block  
12 902 wherein the query is posed: did mismatch occur on the master node to disk(s)  
13 combination or on a slave node to disk(s) combination. (Again, the term slave node in  
14 this instance includes the originator node to allow all nodes to be polled.)  
15

16 If the mismatch is on a slave node, the algorithmic process moves to yet another  
17 decision block, block 903, wherein the query is posed: is mismatch due to missing file(s)  
18 from, or due to extra file(s) on, this particular slave node's physical disks? If due to a  
19 missing file(s) from the disk, the process moves to block 906 wherein the missing file(s)  
20 is retrieved from the originator node associated with that missing file (the originator node  
21 being that node through which the missing file was first introduced into the network), or  
22 retrieved from the master node whereupon the algorithmic process stops. This retrieval is  
23 performed by the particular slave node's file replication service in the manner described



1 a backup capacity. As each slave node acquires the updated file from the originator or  
2 master node as earlier described, that slave node can serve as another source of backup  
3 since it now has the new file. After that slave node acquires the new file, its directory  
4 provider service can secondarily publish (re-publish) the new file's FVV from its global  
5 workspace. There may be an additional tag attached to that secondary publication  
6 advising that it is not the originator or master node but it can be used by yet other slave  
7 nodes in a further backup capacity, but only if the both the originator and master nodes  
8 are not available to serve up the new file. Or, it can be available for that purpose even if  
9 originator and master nodes are available and operative. This procedure can be  
10 cumulative in the sense that as each additional slave node acquires the new file from the  
11 originator, master, or other prior "backup-slave" node, it can join the ranks of backup-  
12 slave nodes by also re-publishing the new file's FVV thereby advising that it too can be  
13 used for backup under the same conditions as prior backup slave nodes. This may be a  
14 complicated algorithm, but is one that is implementable and offers a further reliability  
15 advantage of having more than two sources of the new data file.

16  
17 A second alternative embodiment is an enhancement primarily for large files.  
18 Embodiments of the present invention currently can send file content via Common  
19 Information Model/eXtensible Markup Language (CIM/XML). Because CIM/XML  
20 cannot handle sending and receiving binary content, 64 bit encoding is used. The content  
21 thus needs to be "chunked"(chunk size to be determined) as all of the content must be  
22 parsed by the remote server before processing the request and it is undesirable to read and  
23 maintain all of the content in memory. A specialized use of CIM/XML may achieve the



1 same result as the following alternative embodiment, such as, for example, the server  
2 processing the request and not parsing it entirely, but this requires specialized code in the  
3 generic request path. The alternative embodiment is to use a "PUT" command with  
4 Hyper Text Transfer Protocol (HTTP), the combination termed "HTTP PUT", to deliver  
5 the binary content to the server. The HTTP PUT command allows one to use one of the  
6 services of the Common Information Model Object Manager (CIMOM) to bypass the  
7 parser completely. The server would post the large file to the CIMOM which would save  
8 the large file in a scratch space. Then, the server would send a quick XML note to tell  
9 the client where the file came from and what it should be saved as. This eliminates the  
10 need to encode the binary content, and the content will automatically be chunked by  
11 Transmission Control Protocol/Internet Protocol (TCP/IP). Each chunk can then be read  
12 and written directly to disk as it is received. Information regarding length and content of  
13 the file can be placed in a header that resides at the beginning of the file. This may  
14 include the name, version, length, and a hash value representing the file's checksum.  
15 When the transfer and all checks on the file are complete, a status can be returned via the  
16 HTTP PUT response to the client. Non-standard http headers can be included as well to  
17 indicate the operation's status if a simple status is not sufficient. The PUT may also  
18 include security information, such as a message digest security algorithm, for example,  
19 an MD2 message digest, which can be authenticated by the server.

20  
21 When the file is transferred, the webserver or CIMOM must have a way of  
22 knowing that the file is intended to be replicated in accordance with the principles of the  
23 present invention. This can be achieved through the use of a Uniform Resource

1 Identifier (URI) which indicates that the HTTP PUT operation is intended for a  
2 specialized resource on the server (i.e. file replication or “persistence”). As an example,  
3 the URI CIMOM is used to indicate that POST requests are intended for the CIMOM  
4 component of the webserver, POST being another command similar to PUT.

5  
6 The steps for replicating the file would then be as follows:

- 7 1. Client sends the file using HTTP POST. The URI that is specified is known to  
8 both client and server as an indicator to the server that this PUT request is  
9 intended to be replicated in accordance with the principles of the present  
10 invention.
- 11 2. Server receives the pieces and places the file into file replication service  
12 storage.
- 13 3. The server checks validity of the header, then uses that validity information to  
14 both ensure that the entire file transferred and to name the file appropriately.
- 15 4. The webserver/CIMOM then makes an internal request to the file replication  
16 service, thereby advising a file has arrived to be processed.
- 17 5. The module processes the request and indicates successful or failed delivery  
18 back to the client.

19  
20 The present invention may thus be embodied in many different forms, including,  
21 but not limited to, computer program logic for use with any kind of processor,  
22 programmable logic for use with any kind of programmable logic device, discrete  
23 components, integrated circuitry including application specific integrated circuits

1 (ASICs), or any other means including any combination thereof. Computer program  
2 logic implementing all or part of the functionality described herein may be embodied in  
3 various forms, including, but not limited to, source code form, computer executable form,  
4 and various intermediate forms (e.g. forms generated by an assembler, compiler, linker,  
5 or locator.) Source code may include a series of computer program instructions  
6 implemented in any of various programming languages for use with various operating  
7 systems or operating environments. The source code may define and use various data  
8 structures and communication messages. The source code may be in computer  
9 executable form, or it may be in a form convertible into computer executable form. The  
10 computer program may be fixed in any form either permanently or transitorily in a  
11 tangible storage medium, such as a semiconductor memory device, a magnetic memory  
12 device, an optical memory device, a PC card, or other memory device. The computer  
13 program may be fixed in any form in a signal that is transmittable to a computer using  
14 any of various communication technologies including, but not limited to, analog, digital,  
15 optical, wireless, networking, and internetworking technologies. The computer program  
16 may be distributed in any form as a removable storage medium with accompanying  
17 printed or electronic documentation, preloaded with a computer system (e.g. on system  
18 ROM or fixed disk), or distributed from a server or electronic bulletin board over the  
19 communication system (e.g., the Internet or World Wide Web).

20  
21 The present embodiments are to be considered in all respects as illustrative and  
22 not restrictive. The scope of the invention is indicated by the appended claims rather than

1 by the foregoing description, and all changes which come within the meaning and range  
2 of equivalency of the claims are therefore intended to be embraced therein.

3

44